



# Pimp my Java

Daniel Petisme  
Philippe Charrière



Us?

# @k33g\_org

*aka. Golo Developer Advocate*



@danielpetisme

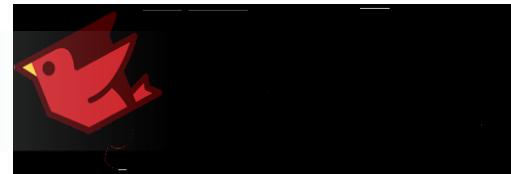


# invokedynamic?

# Java VM

**Java VM**

# Multi-Language VM



*The Java SE 7 platform enables **non-Java languages** to exploit the infrastructure and potential performance optimizations of the JVM.*

*The key mechanism is the **invokedynamic** instruction, which **simplifies the implementation** of compilers and runtime systems for **dynamically typed languages** on the JVM.*

<http://docs.oracle.com/javase/7/docs/technotes/guides/vm/multiple-language-support.html>

# Static

```
public class Adder {  
  
    public Integer add(Integer a, Integer b) {  
        return a + b;  
    }  
  
    public String add(String a, String b) {  
        return a + b;  
    }  
  
    public static void main(String[] args) {  
        Adder myAdder = new Adder();  
        int x = 10;  
        int y = 10;  
  
        List<String> jugs = new ArrayList<>();  
        String theBestOne = "LavaJUG";  
  
        jugs.add(theBestOne);  
        myAdder.add(x, y);  
  
        myAdder.add("The Best JUG is: ", theBestOne);  
    }  
}
```

*Types checking at compilation...*

# Dynamic

```
class Adder {  
  
    def add(a, b) {  
        return a + b  
    }  
  
    def main(args) {  
        def myAdder = new Adder()  
        def x = 10  
        def y = 10  
  
        def jugs = []  
        def theBestOne = "LavaJUG";  
        jugs.add(theBestOne)  
  
        myAdder.add(x, y)  
        myAdder.add("The Best JUG is: ",  
theBestOne)  
  
    }  
}
```

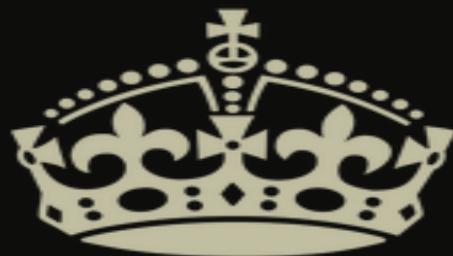
*Types checking at runtime...*

And what's the  
problem?

*Dynamic language compilation*



```
public class Adder {  
  
    public Integer add(Integer a, Integer b) {  
        return a + b;  
    }  
  
    public String add(String a, String b) {  
        return a + b;  
    }  
  
    public static void main(String[] args) {  
        Adder myAdder = new Adder();  
        int x = 10;  
        int y = 10;  
  
        List<String> jugs = new ArrayList<>();  
        String theBestOne = "LavaJUG";  
  
        jugs.add(theBestOne);  
        myAdder.add(x, y);  
  
        myAdder.add("The Best JUG is: ", theBestOne);  
    }  
}
```



KEEP  
CALM  
AND  
COMPILE  
PLEASE



```
public static void main(java.lang.String[]);
  flags: ACC_PUBLIC, ACC_STATIC
  Code:
    stack=3, locals=6, args_size=1
      0: new           #8          // class lavajug/sample1/Adder
      3: dup
      4: invokespecial #9        // Method<init>: ()V
      7: astore_1
      8: bipush        10
     10: istore_2
     11: bipush        10
     13: istore_3
     14: new           #10         // class java/util/ArrayList
     17: dup
     18: invokespecial #11        // Methodjava/util/ArrayList.<init>: ()V
     21: astore        4
     23: ldc            #12         // String LavaJUG
     25: astore        5
     27: aload        4
     29: aload        5
     31: invokeinterface #13,  2   // InterfaceMethodjava/util/List.add: (Ljava/lang/Object;)Z
     36: pop
     37: aload_1
     38: iload_2
     39: invokestatic  #3          // Methodjava/lang/Integer.valueOf: (I)Ljava/lang/Integer;
     42: iload_3
     43: invokestatic  #3          // Methodjava/lang/Integer.valueOf: (I)Ljava/lang/Integer;
     46: invokevirtual #14        // Methodadd: (Ljava/lang/Integer;Ljava/lang/Integer;)Ljava/lang/Integer;
     49: pop
     50: aload_1
     51: ldc            #15         // String The Best JUG is:
     53: aload        5
     55: invokevirtual #16        // Methodadd: (Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
     58: pop
     59: return
```



```
public static void main(java.lang.String[]);
```

```
  flags ACC_PUBLIC ACC_STATIC
```

```
Code:
```

```
stack=3, locals=6, args_size=1
 0: new           #8          // class lavajug/sample1/Adder
 3: dup
 4: invokespecial #9        // Method<init>: ()V
 7: astore_1
 8: bipush        10
10: istore_2
11: bipush        10
13: istore_3
14: new           #10         // class java/util/ArrayList
17: dup
18: invokespecial #11       // Methodjava/util/ArrayList.<init>: ()V
21: astore        4
23: ldc            #12         // String LavaJUG
25: astore        5
27: aload        4
29: aload        5
31: invokeinterface #13,  2   // InterfaceMethodjava/util/List.add: (Ljava/lang/Object;)Z
36: pop
37: aload_1
38: iload_2
39: invokestatic  #3          // Methodjava/lang/Integer.valueOf: (I)Ljava/lang/Integer;
42: iload_3
43: invokestatic  #3          // Methodjava/lang/Integer.valueOf: (I)Ljava/lang/Integer;
46: invokevirtual #14        // Methodadd: (Ljava/lang/Integer;Ljava/lang/Integer;)Ljava/lang/Integer;
49: pop
50: aload_1
51: ldc            #15         // String The Best JUG is:
53: aload        5
55: invokevirtual #16        // Methodadd: (Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
58: pop
59: return
```

# 1-Types present at the callsites



```
public static void main(java.lang.String[]);
```

```
  flags ACC_PUBLIC ACC_STATIC
```

```
Code:
```

```
stack=3, locals=6, args_size=1
 0: new           #8          // class lavajug/sample1/Adder
 3: dup
 4: invokespecial #9        // Method<init>: ()V
 7: astore_1
 8: bipush        10
10: istore_2
11: bipush        10
13: istore_3
14: new           #10         // class java/util/ArrayList
17: dup
18: invokespecial #11       // Methodjava/util/ArrayList.<init>: ()V
21: astore        4
23: ldc            #12         // String LavaJUG
24: astore        5
27: aload         4
29: aload         5
31: invokeinterface #13,  2   // InterfaceMethodjava/util/List.add: (Ljava/lang/Object;)Z
36: pop
37: aload_1
38: iload_2
39: invokestatic  #3          // Methodjava/lang/Integer.valueOf: (I)Ljava/lang/Integer;
42: iload_3
43: invokestatic  #3          // Methodjava/lang/Integer.valueOf: (I)Ljava/lang/Integer;
46: invokevirtual #14        // Methodadd: (Ljava/lang/Integer;Ljava/lang/Integer;)Ljava/lang/Integer;
49: pop
50: aload_1
51: ldc            #15         // String The Best JUG is:
53: aload         5
55: invokevirtual #16        // Methodadd: (Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
58: pop
59: return
```

## 2- Method must exist at compile time



```
public static void main(java.lang.String[]);
```

```
  flags ACC_PUBLIC ACC_STATIC
```

```
Code:
```

```
stack=3, locals=6, args_size=1
 0: new           #8          // class lavajug/sample1/Adder
 3: dup
 4: invokespecial #9        // Method<init>: ()V
 7: astore_1
 8: bipush        10
10: istore_2
11: bipush        10
13: istore_3
14: new           #10         // class java/util/ArrayList
17: dup
18: invokespecial #11       // Methodjava/util/ArrayList.<init>: ()V
21: astore        4
23: ldc            #12         // String LavaJUG
24: astore        5
27: aload         4
29: aload         5
31: invokeinterface #13,  2   // InterfaceMethodjava/util/List.add: (Ljava/lang/Object;)Z
36: pop
37: aload_1
38: iload_2
39: invokestatic  #3          // Methodjava/lang/Integer.valueOf: (I)Ljava/lang/Integer;
42: iload_3
43: invokestatic  #3          // Methodjava/lang/Integer.valueOf: (I)Ljava/lang/Integer;
46: invokevirtual #14        // Methodadd: (Ljava/lang/Integer;Ljava/lang/Integer;)Ljava/lang/Integer;
49: pop
50: aload_1
51: ldc            #15         // String The Best JUG is:
53: aload         5
55: invokevirtual #16        // Methodadd: (Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
58: pop
59: return
```

## 2- Method must exist at compile time

## 3- No relinking possible

# How to invoke

## invokeStatic

```
Integer.valueOf(10);  
invokestatic #3 // Method java/lang/Integer.valueOf: (I)Ljava/lang/Integer;
```

## invokeSpecial

```
new ArrayList();  
invokespecial #11 // Method java/util/ArrayList."<init>": ()V
```

## invokeInterface

```
jugs.add(theBestOne);  
invokeinterface #13, 2 // InterfaceMethod java/util/List.add: (Ljava/lang/Object;)Z
```

## invokeVirtual

```
myAdder.add("The Best JUG is: ",theBestOne);  
invokevirtual #16 // Method lavajug/sample1/java/Adder.add: (Ljava/lang/String; Ljava/lang/String;)Ljava/lang/String;
```

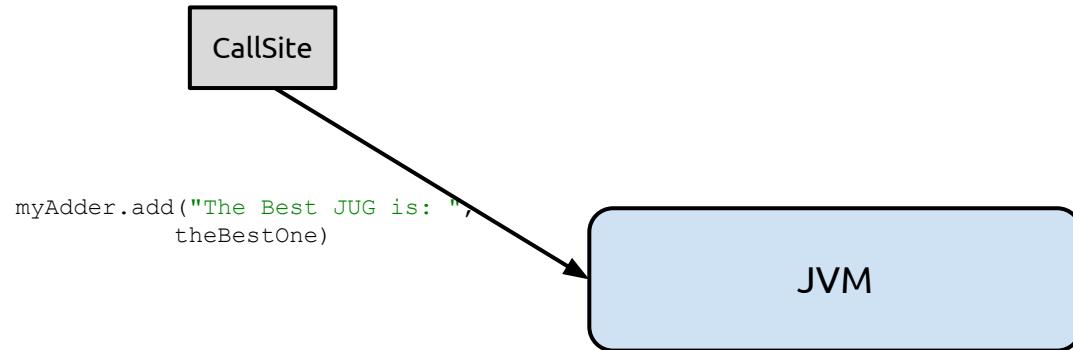
## CallSite

Receiver	ReceiverClass	Name	Descriptor
myAdder	Adder	add	(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;

CallSite

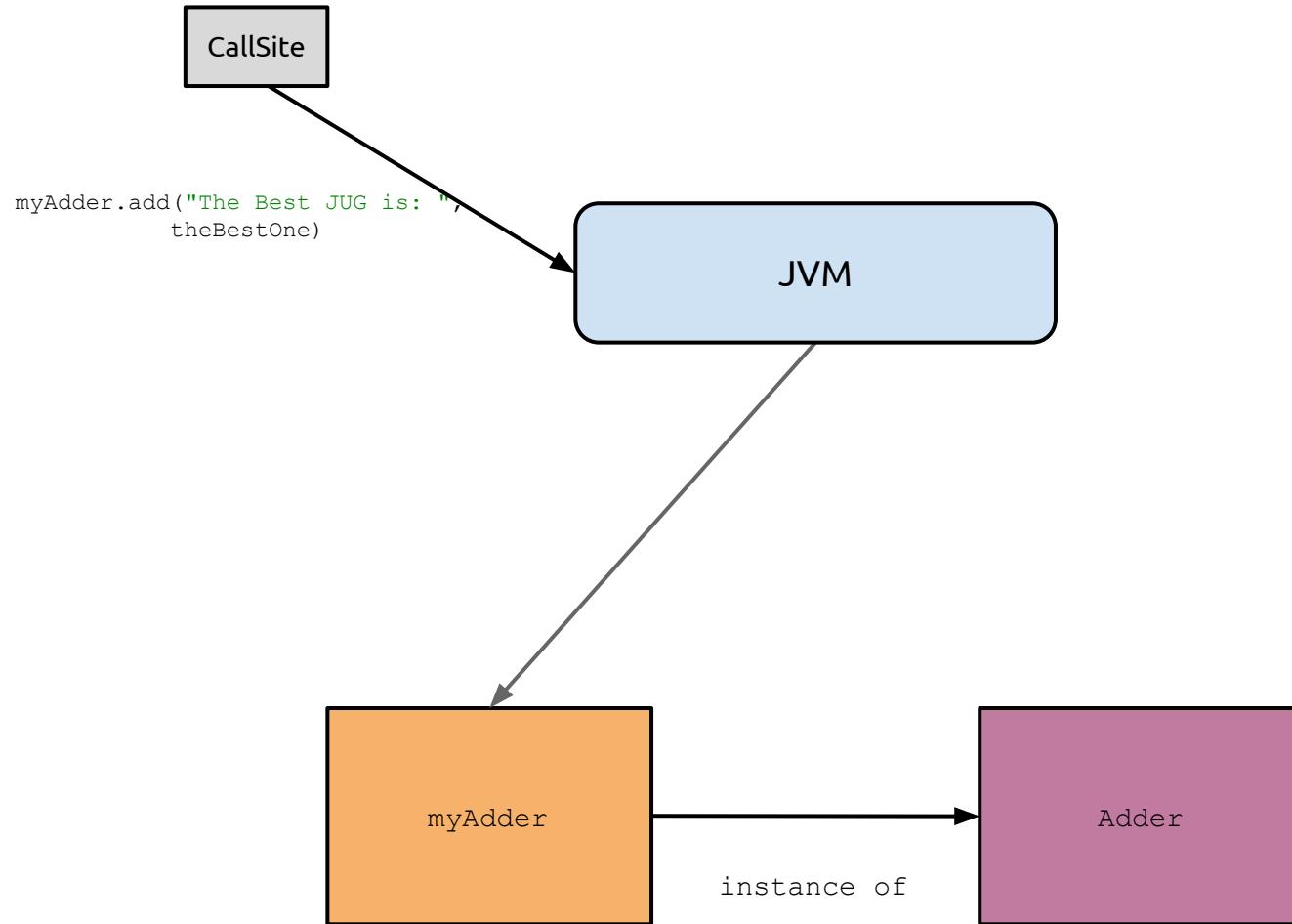
## CallSite

Receiver	ReceiverClass	Name	Descriptor
myAdder	Adder	add	(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;



## CallSite

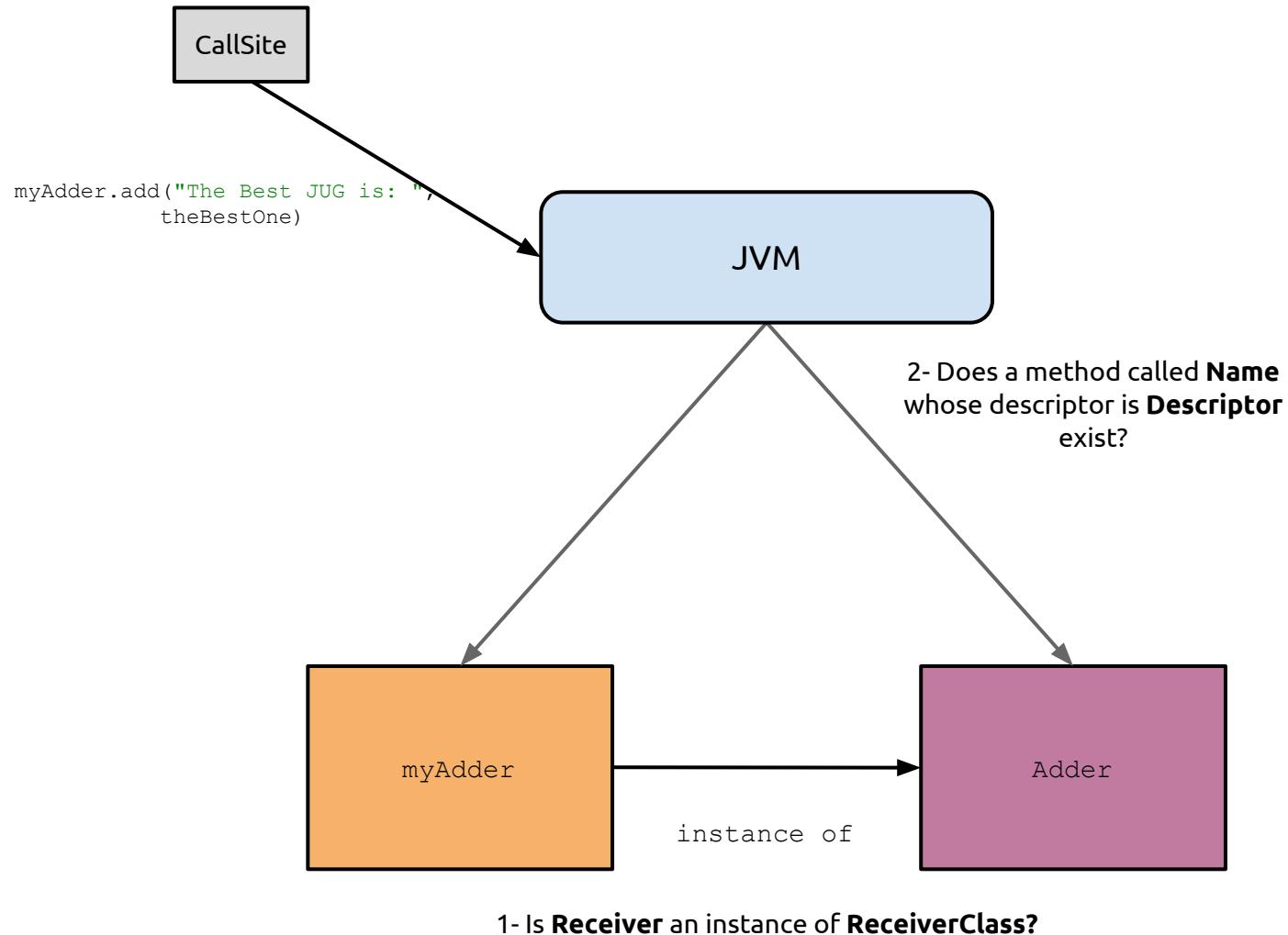
Receiver	ReceiverClass	Name	Descriptor
myAdder	Adder	add	(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;



1- Is **Receiver** an instance of **ReceiverClass**?

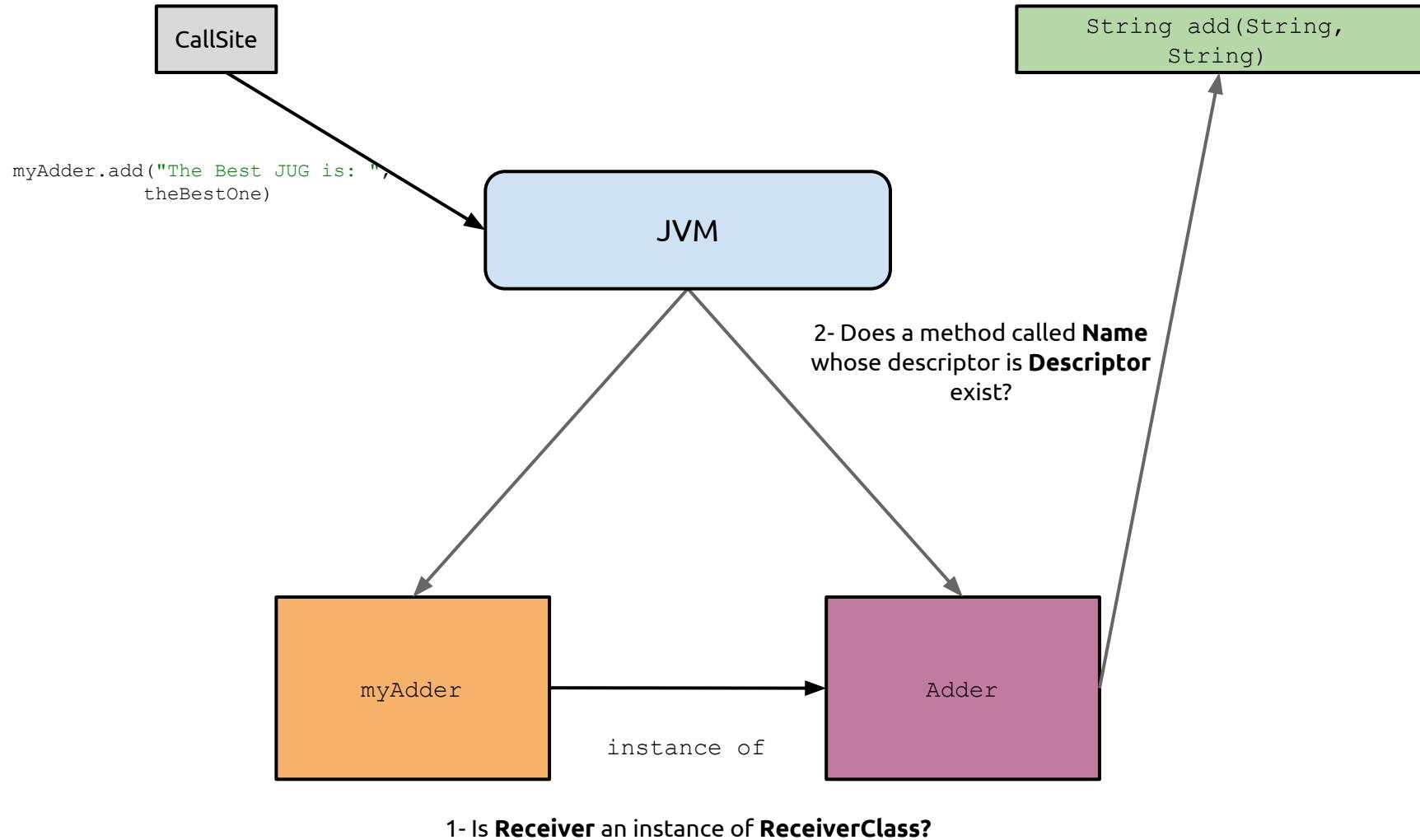
## CallSite

Receiver	ReceiverClass	Name	Descriptor
myAdder	Adder	add	(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;



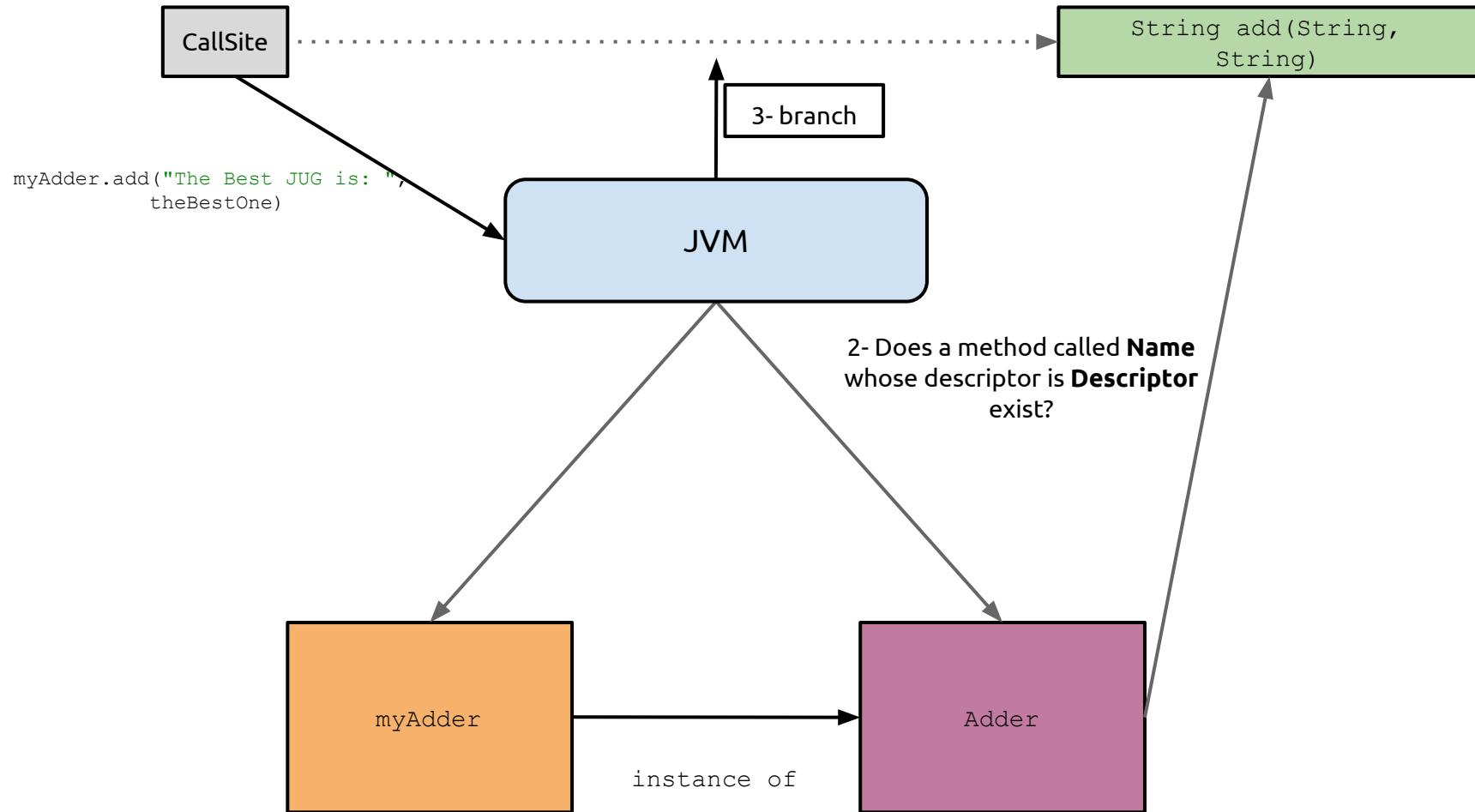
## CallSite

Receiver	ReceiverClass	Name	Descriptor
myAdder	Adder	add	(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;



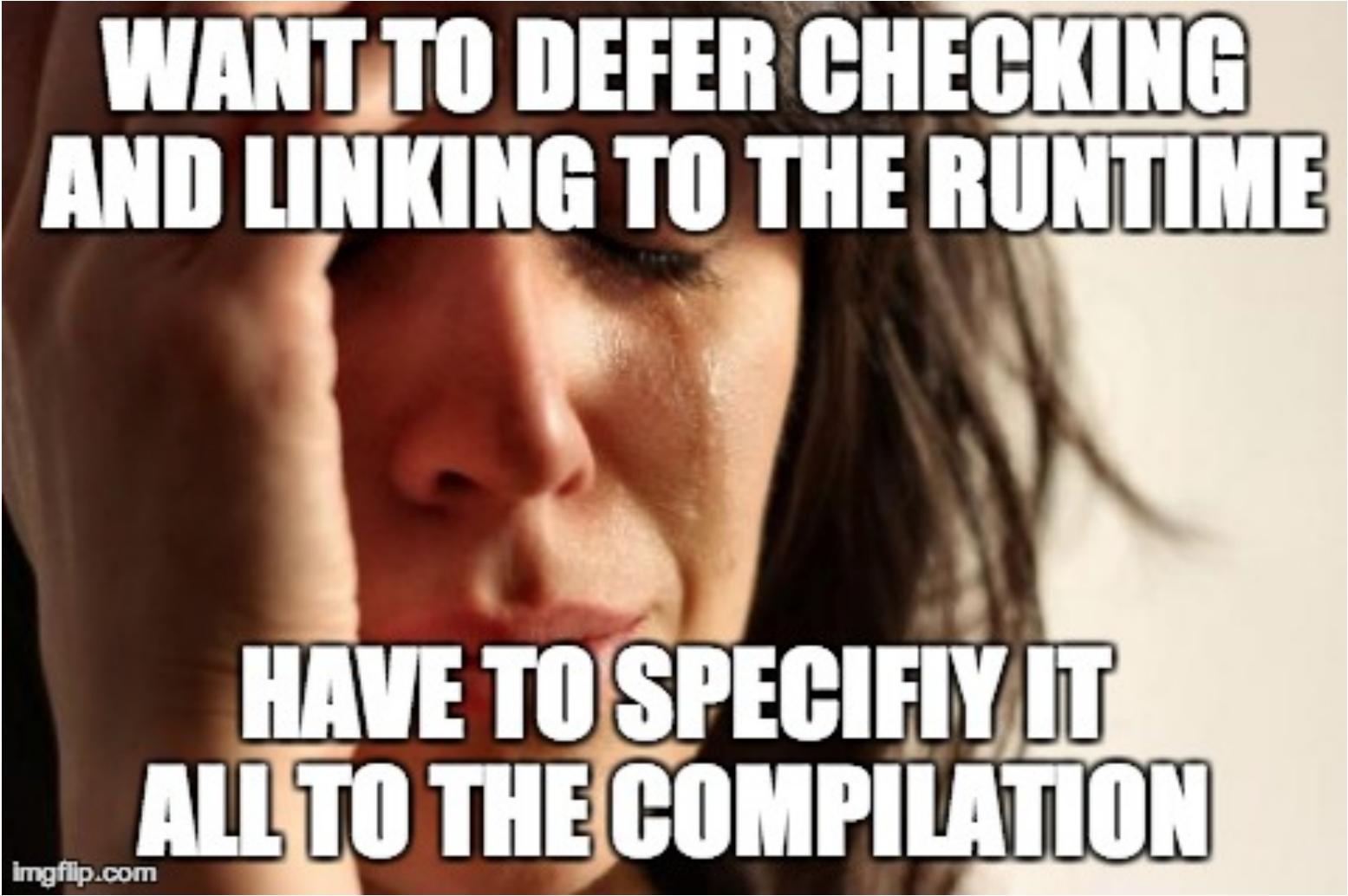
## CallSite

Receiver	ReceiverClass	Name	Descriptor
myAdder	Adder	add	(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;



1- Is **Receiver** an instance of **ReceiverClass**?

# Dynamic languages

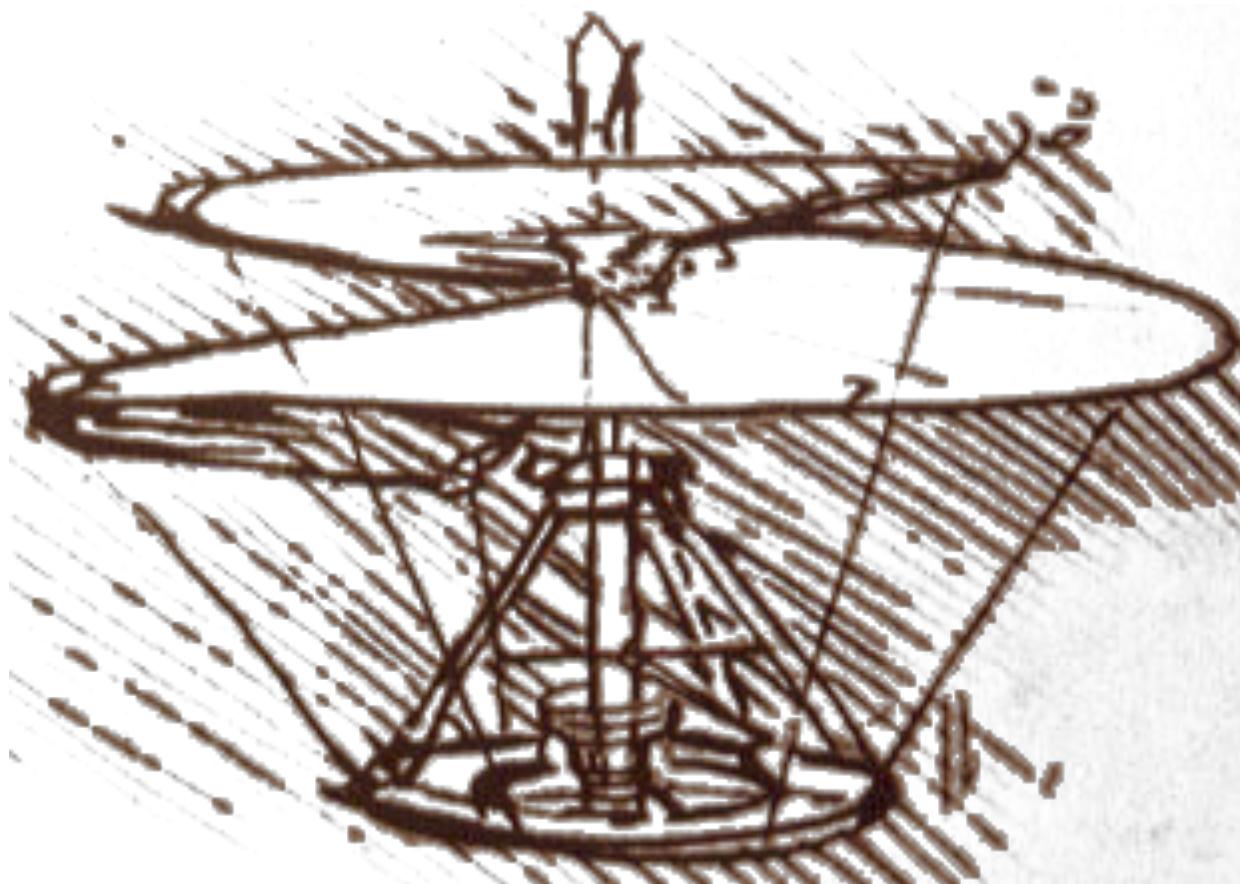


**WANT TO DEFER CHECKING  
AND LINKING TO THE RUNTIME**

**HAVE TO SPECIFY IT  
ALL TO THE COMPIRATION**

invokedynamic!

# *JSR-292: Da Vinci Machine*



New bytecode: `invokedynamic`

A new  
invokedynamic  
bytecode, that's  
all?

# invokedynamic

*Use symbolic specifications at compilation  
Defer callsite bindings to runtime*

# invoke dynamic

*user-defined  
bytecode*

*method linking  
& adaptation*

*Bonus: JVM Optimizations*

*New!*

java.lang.invoke

# Chapter I

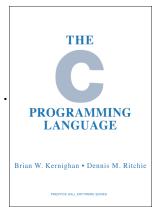
# Method Handles

# Method Handles?

# Method Handles



```
int (*functionPtr)(int, int);
```



```
String word = "Java";
```

MethodHandle Factory embeds method handle access restrictions

```
MethodHandles.Lookup lookup = MethodHandles.lookup();
```

```
Class receiverType = String.class;
```

```
Class returnType = char.class;
```

```
Class paramType = int.class;
```

Seek for a public exposed Method

```
MethodHandle charAtHandle = lookup.findVirtual(
```

receiverType,  
"charAt",

Where to look  
for the Method

```
MethodType.methodType(returnType, paramType)
```

```
);
```

Descriptor

```
char charAt0 = (char) charAtHandle.invokeWithArguments(word, 0);
```

```
assert charAt0 == 'J';
```

Invocation with the receiver instance and the parameter(s)

Method Handles  
Vs.  
Reflection?

# Method Handles Vs. Reflection?

*Better/Easier discovery mechanism*

*Faster + Combination facilities*

# Method Handles

Vs.

# Reflection?

*Better/Easier discovery mechanism*

Method Handles  
&  
Reflection!

```
Lookup lookup = MethodHandles.lookup();

Method formatMethod = MessageFormat.class.getMethod(
    "format",
    String.class,
    Object[].class
);

MethodHandle greeterHandle = lookup.unreflect(formatMethod);

assert greeterHandle.type() == methodType(
    String.class,
    String.class,
    Object[].class
);
```

From `java.lang.reflect` to `java.lang.invoke`

Looks fun?  
You ain't see nothing!

```

Lookup lookup = MethodHandles.lookup();

Method formatMethod = MessageFormat. class.getMethod(
    "format", String. class, Object[]. class );

MethodHandle greeterHandle = lookup. unreflect(formatMethod);

greeterHandle = greeterHandle.
    bindTo("Hello {0} {1}{2}").
    asCollector(Object[]. class, 3);
    
```

*//Arguments manipulation*

```

greeterHandle = MethodHandles. insertArguments(1,
    greeterHandle,
    greeterHandle.type().parameterCount() - 1,
    "!");
    
```

*//Combining*

```

Method toUpperMethod = String. class.getMethod("toUpperCase");
MethodHandle toUpperHandle = lookup.unreflect(toUpperMethod);

greeterHandle = MethodHandles. filterReturnValue(greeterHandle,
    toUpperHandle);
    
```

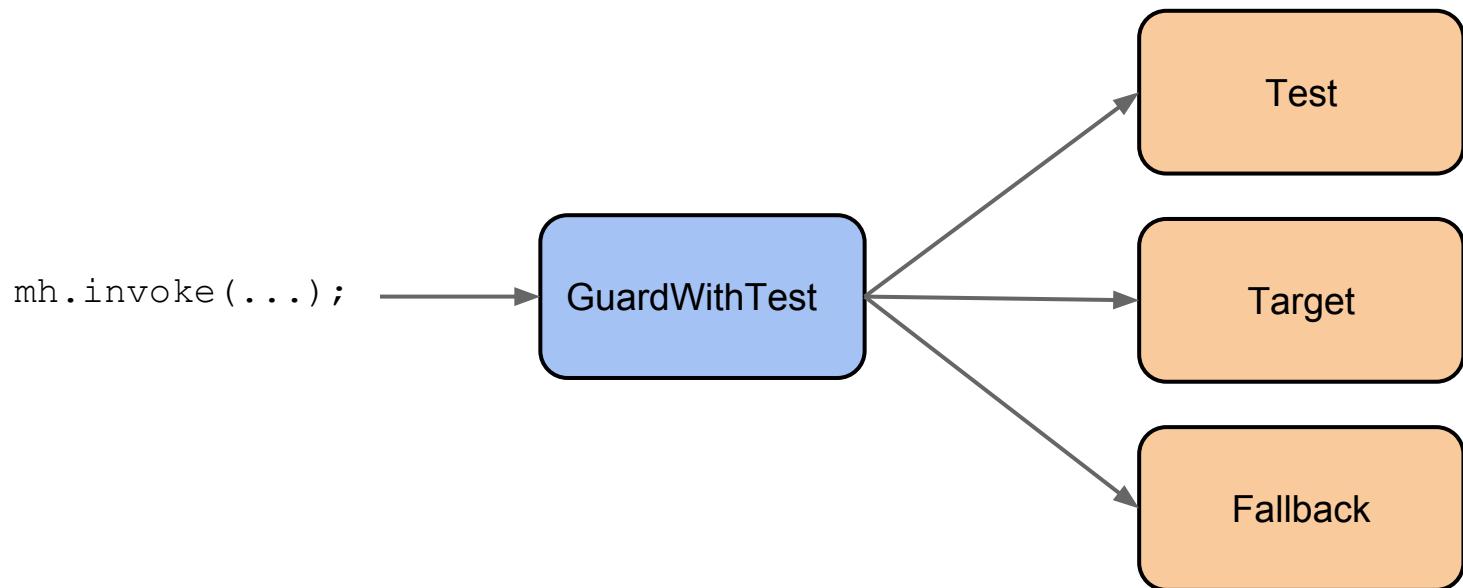
*//Test*

```

String greet = (String) greeterHandle.invokeWithArguments("John", "Doe");
assert greet.equals("HELLO JOHN DOE!");
    
```

The diagram features several callout boxes with arrows pointing to specific parts of the code:

- A blue box labeled "Curry power ❤️" points to the line `greeterHandle = greeterHandle.bindTo("Hello {0} {1}{2}").`
- A blue box labeled "Now representing a 3 args method" points to the line `greeterHandle = MethodHandles. insertArguments(1,`
- A blue box labeled "Add/remove/move arguments" points to the line `greeterHandle.type().parameterCount() - 1,`
- A blue box labeled "Split the concerns, then combine the methods" points to the line `greeterHandle = MethodHandles. filterReturnValue(greeterHandle,`



*Get prepared for  $\lambda$ ...*

# Chapter II

# BootStrapping

**DynamicInvokerClass**

```
invokeDynamic symMethodName:  
            symMethodType
```

## At compilation: Method pointers

**ActualImplementationClass**

```
actualMethod(actualType)
```

**DynamicInvokerClass**

```
invokeDynamic symMethodName:  
    symMethodType
```

# How to??

**ActualImplementationClass**

```
actualMethod(actualType)
```

# Delegation to a BootStrap Method

## **DynamicInvokerClass**

```
invokeDynamic symMethodName:  
    symMethodType
```

*Each InvokeDynamic instruction refers to a  
**BootStrap Method invoked the 1st time***

**BootStrap Method (BSM)**

## **ActualImplementationClass**

```
actualMethod(actualType)
```

## **DynamicInvokerClass**

```
invokeDynamic symMethodName:  
    symMethodType
```

## **BootStrap Method (BSM)**

*lookup*

## **ActualImplementationClass**

```
actualMethod(actualType)
```

### **DynamicInvokerClass**

```
invokeDynamic symMethodName:  
    symMethodType
```

**BootStrap Method (BSM)**

MethodHandle(s)

*Compute the MH  
representing the  
target*

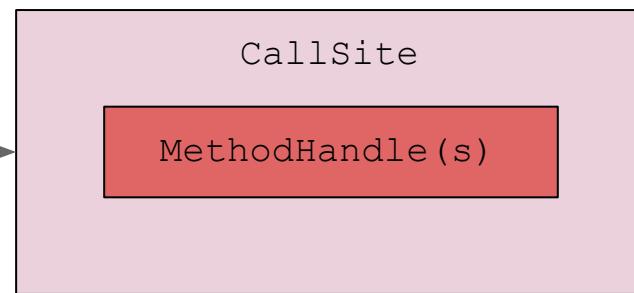
### **ActualImplementationClass**

```
actualMethod(actualType)
```

## **DynamicInvokerClass**

```
invokeDynamic symMethodName:  
    symMethodType
```

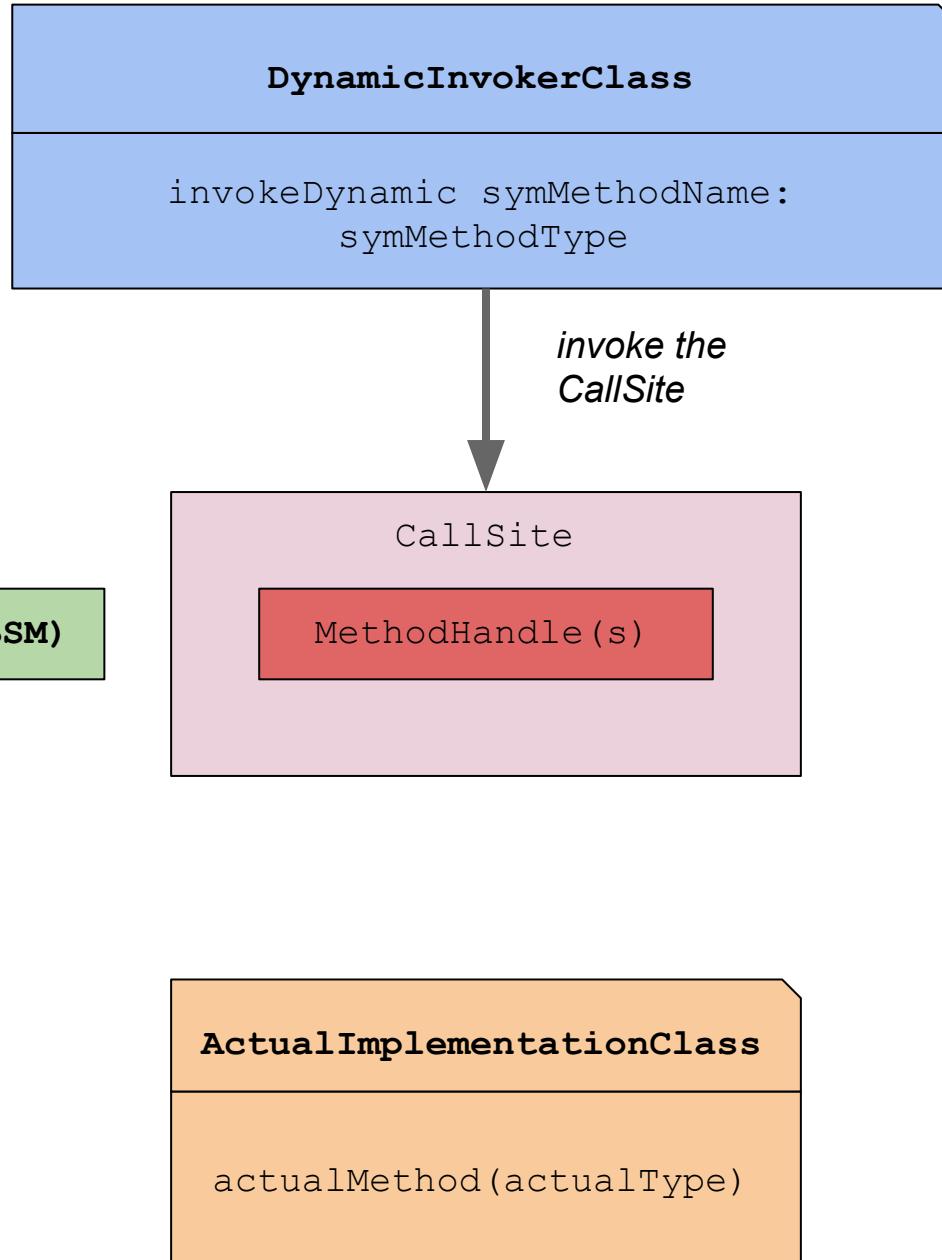
**BootStrap Method (BSM)**

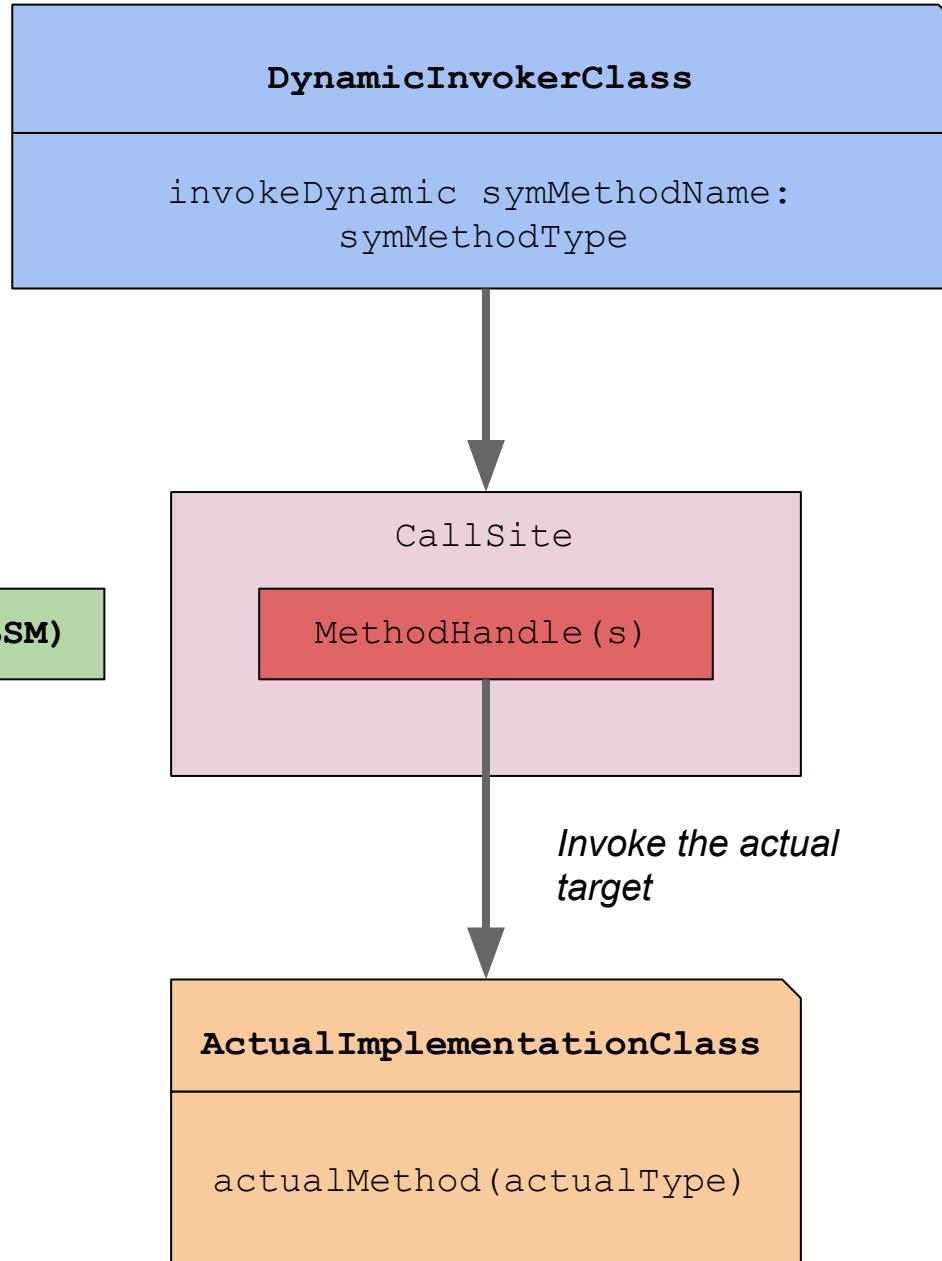


*Wrap the MH  
representation in a  
CallSite and return it*

## **ActualImplementationClass**

```
actualMethod(actualType)
```





# BootStrap Method needs?

# BootStrap Method needs:

The invocation context (`MethodHandles.Lookup`)

A symbolic name (`String`)

The resolved type descriptor of the call (`MethodType`)

Optional arguments (constants)

```
public static CallSite bootstrap(Lookup lookup, String name, MethodType
type)
throws Throwable {

MethodHandle target = lookup.findStatic(MessageFormat. class, "format",
methodType(String. class, String.class, Object[]. class))
.bindTo("Hello {0} {1}!")
.asCollector(String[]. class, 2)
.asType(type);

return new ConstantCallSite(target);
}

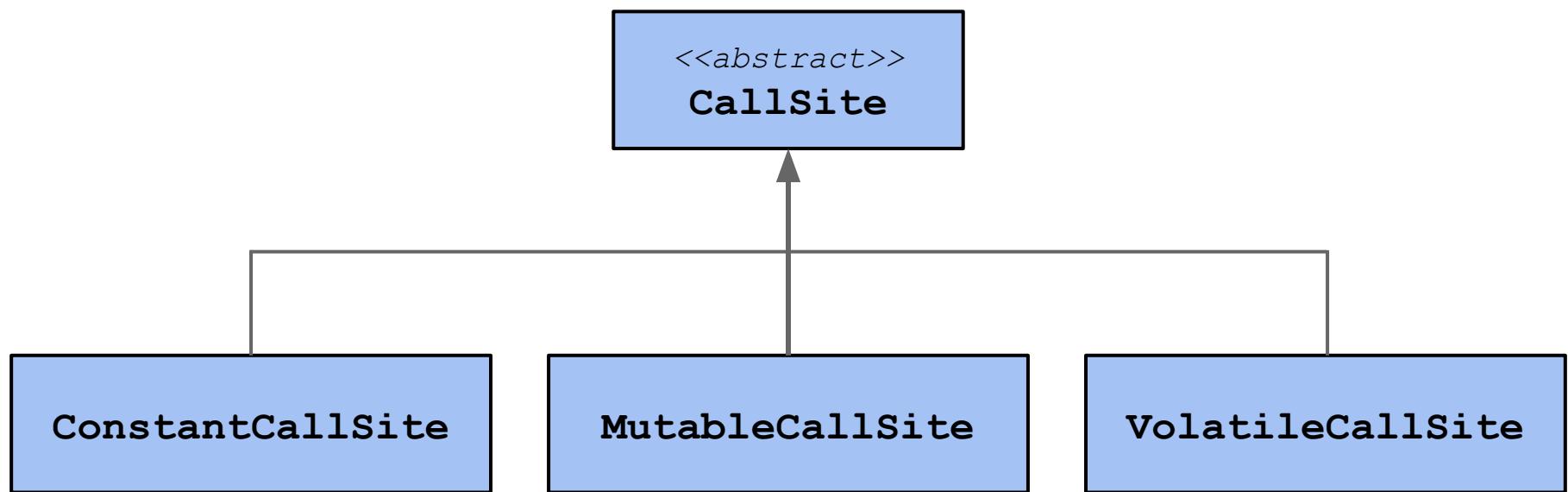
public static void main(String[] args) throws Throwable{
CallSite callSite = bootstrap(
MethodHandles.lookup(),
"myGreeter",
methodType(String. class, String.class, String.class)
);

String greet = (String) callSite.dynamicInvoker().invokeWithArguments(
"John", "Doe"
);

assert greet.equals("Hello John Doe!");
}
```

The target representation

The binding between invoker & receiver at runtime



# Chapter III

# Bytecode strikes back

How to produce  
invokeDynamic  
bytecode ops?

You can't!

You can't!  
*at the moment...*



<http://asm.ow2.org/>

All together: YAAL

YAAL = Yet Another Awesome Language

= ~~Makes me rich~~

= ~~Makes me famous~~

= ~~Useful~~

= Kind of invokedynamic how to

# Demo time!

<https://github.com/danielpetisme/pimp-my-java>

~~Plan A~~

Plan B



```
greet John Doe
```

# Yaal Syntax

An operation



```
greet John Doe
```

# Yaal Syntax

An operation

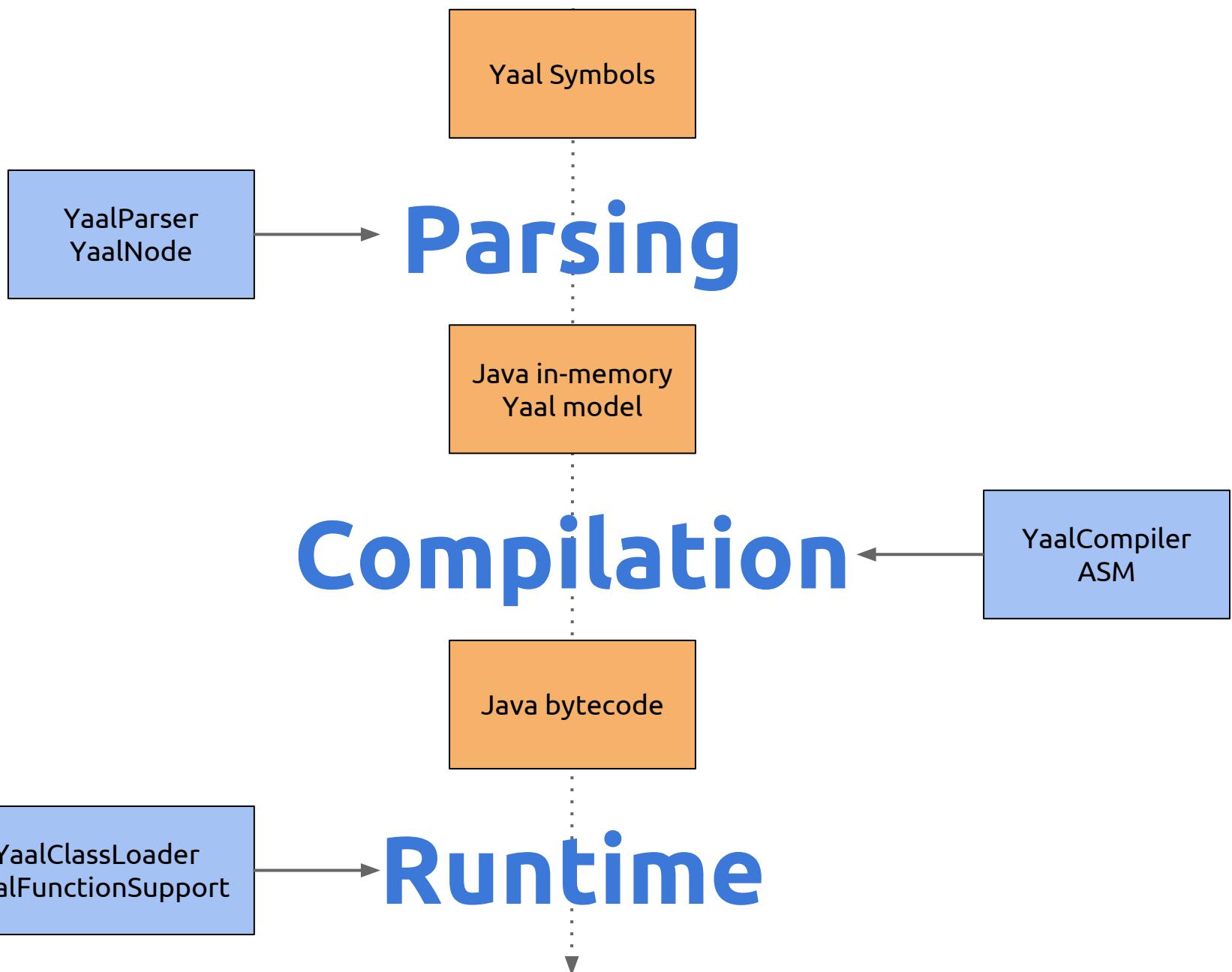


greet John Doe



N arguments

# Yaal Syntax



```

MethodVisitor mv = writer.visitMethod(
    ACC_STATIC | ACC_PUBLIC, "main", "([Ljava/lang/String;)V", null,
    null);

for (YaalNode node : nodes) {

    for(String argument : node.arguments) {
        mv.visitLdcInsn(argument);
    }
}

String bootstrapOwner = "lavajug/yaal/runtime/YaalFunctionSupport";
String bootstrapMethod = "bootstrap";
String desc = MethodType.methodType(
    CallSite.class,
    MethodHandles.Lookup.class,
    String.class,
    MethodType.class).toMethodDescriptorString();

Handle bsm = new Handle(
    H_INVOKESTATIC,
    bootstrapOwner,
    bootstrapMethod,
    desc
);

mv.visitInvokeDynamicInsn(
    node.operator,
    genericMethodType(node.arguments.length()),
    toMethodDescriptorString(),
    bsm
);
}

```

**Creating a main**

**push args on the stack**

**BSM definition**

**Creating an invokedynamic instruction**

# YaalCompiler#compile

```
public class YaalFunctionSupport {

    public static CallSite bootstrap(
        MethodHandles.Lookup lookup, String name, MethodType type
    ) throws NoSuchMethodException, IllegalAccessException {
        Method predefinedMethod = findMethodByName(Predefined.class, name);

        MethodHandle target = lookup.unreflect(predefinedMethod).asType
            (type);
        return new ConstantCallSite(target);
    }
    ...
}
```

unreflect a method from Yaal  
Predefined functions

To take away

< Java 1.7 Type checking at compilation.  
Hardcoded callsites with types

# Java 1.7

invoke dynamic

*user-defined bytecode*

*Bytecode (ASM) +*

*Bootstrap*

*method linking & adaptation*

*MethodHandle*

*Use symbolic specifications at compilation  
Defer callsite bindings to runtime*

One more thing...

# Thanks @jponge

## *Demystifying invokedynamic*

### Part I

<http://www.oraclejavamagazine-digital.com/javamagazine/20130102?pg=50#pg50>

### Part II

<http://www.oraclejavamagazine-digital.com/javamagazine/20130506?pg=42#pg42>

Golo for newbies!  
-> @k33g\_org



**Le petit langage qui donne des super-pouvoirs**



# Plan

**Golo Origins**  
**Golo Basics**  
**Golo Advanced**  
**Performances (Troll?)**  
**Ecosystème**  
**Golo & moi**

# Golo Origins

Golo? Quoi ?

Nouveau langage

Java 7, 8

**Invokedynamic**

# Doc Ponge

(again)

@jpongē

#gololang  
@golo\_lang  
<http://golo-lang.org>

Sérieux ?  
Encore un  
langage pour la  
JVM !



Golo



**Simple à apprendre**  
**Léger 360 ko**  
**Evolutif très facilement**

Golo? Pourquoi ?

# **Dynamid**

## **Expertise Invokedynamic**

## **Middlewares dynamiques**

**Besoin de Faciliter les  
expérimentations de  
recherche**

1 langage

&

1 runtime

1 **language** > dialectes  
&

1 **runtime** > capacités

**Golo est né  
construit pour Invokedynamic  
prêt à conquérir (tout) le monde**

Golo =  
1 runtime léger  
+ 1 langage simple  
*>> DSL & Expérimentations*

# Golo Basics

# #DEMO

## petit cours “rapide” de Golo ou “Golo par l’exemple”

Golo  
Advanced

# Modifier Golo

**#DEMO**  
**modifier Golo!!!**

# Scripter Java en Golo

# **Utilisation du Golo Class Loader**



[www.sparkjava.com](http://www.sparkjava.com)

jackson

**Google App Engine**  
[goloo-framework.appspot.com](http://goloo-framework.appspot.com)

Troll ... Pas Troll

**Comparons ce qui  
est comparable**

**static <> dynamic**

**Scala : static**  
aussi (+?) **rapide** que Java  
**DSL fonctionnel** gigantesque  
**mais apprentissage très long**

**Groovy : dynamic (\*)**

**moins rapide que Java**

**mais très complet,**

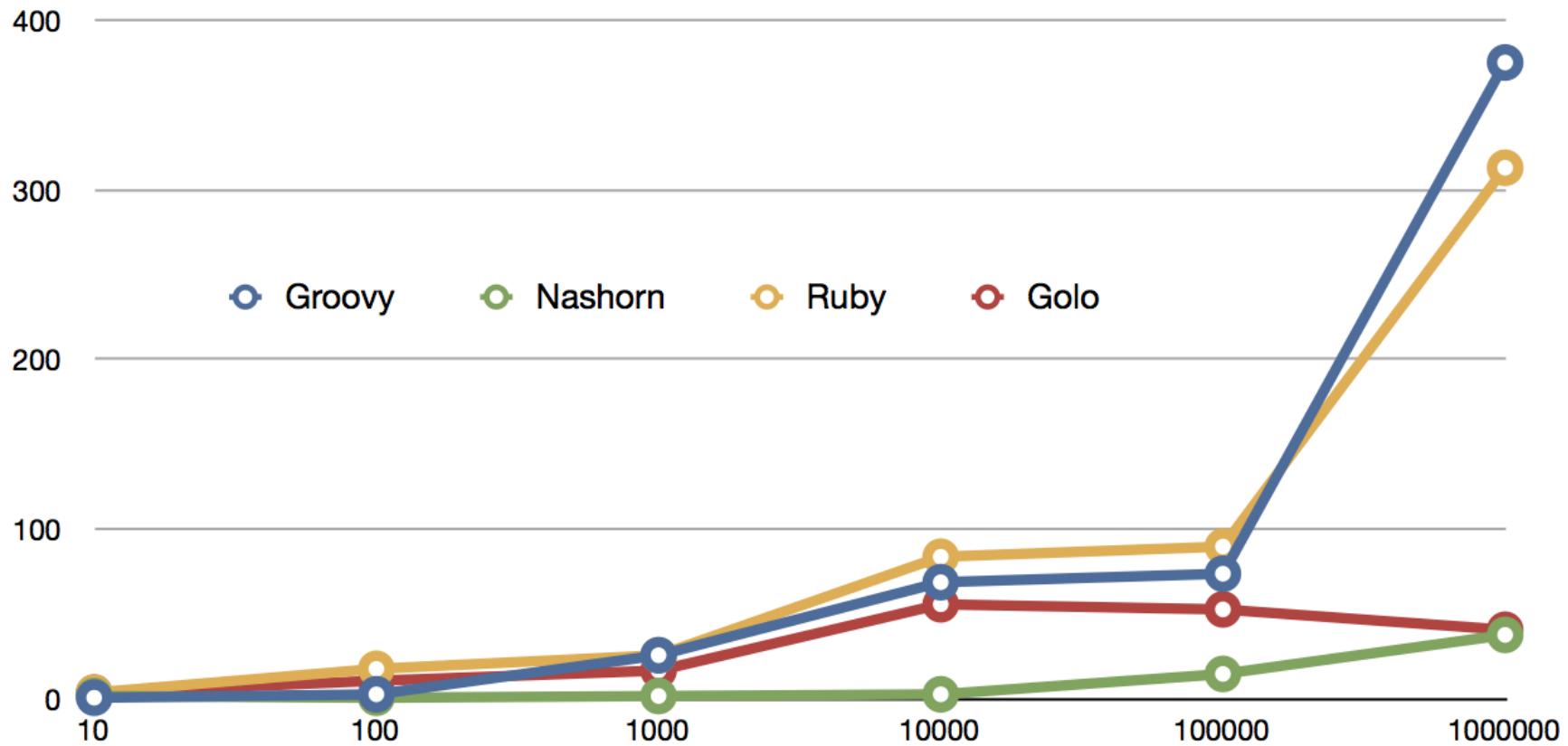
**>> : builders (html, json, ...), ...**

**apprentissage rapide**

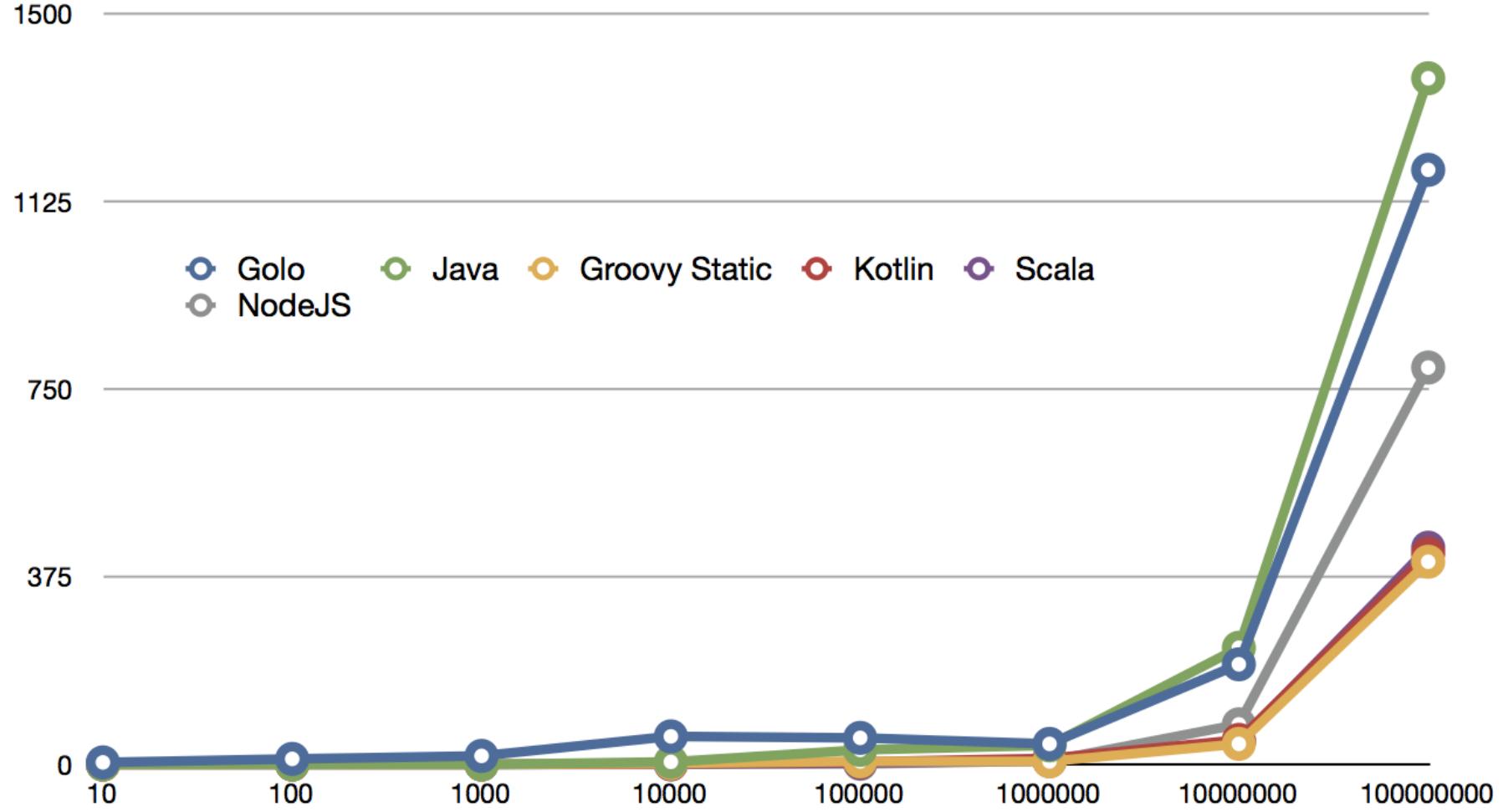
**(\*) : mode statique possible**

**Golo : dynamic**  
**moins rapide que Java,**  
**plus rapide que JRuby,**  
**“Sucrette syntaxique” pour Java &**  
**“goodies”**  
**apprentissage ultra rapide**

# Dynamiques(à relativiser)



# Golo vs Statiques



# Ecosystème Golo

**Projets de recherche**

Congolo / Frédéric Le Mouël

Hardengolo / Nicolas Stouls

**IDE**

Eclipse / Jeff Maury

Netbeans / David Festal

**Tools**

Intégration OSX (homebrew) / Romain

Lespinasse

Intégration Linux / Henri Gomez

**Web**

Intégration Vert.x / Julien Viet

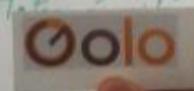
**Futurs évangélistes**

**Daniel Petisme**, Thierry Chantier

**Let us make sure history  
never forgets the name ...**

**Golo**

**JL Picard**



Je l'utilise pour

...

**Apprendre Java  
Prototyper  
DSL Estimations  
DSL G° Exigences  
Expérimentations Acteurs  
Webapps**

Mais il y a  
probablement de  
nombreuses  
possibilités

Drones  
Robots  
Domotique  
Smartgrids

...

**à vous d'  
expérimenter!**

Et après ?

# Concurrency APIs

...

**<https://github.com/k33g/golo-cook-book>**

You have  
Questions

We may have  
Answers